CENTRE DE RECHERCHE
SUR LES TRANSPORTS
(C.R.T.)
CENTRE FOR RESEARCH
ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE : (514) 343-7575
EMAIL : crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX) : (514) 343-7121

92-07824/

CRT-820

A Practical Algorithm for Finding the Largest Clique in a Graph

by

Michael W. Carter[1]
Michel Gendreau[2]

[1] Department of Industrial Engineering, University of Toronto, Toronto, Ontario Canada M5S 1A4.
[2] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montreal (Quebec), Canada H3C 3J7.

Centre de recherche sur les transports — Publication #820

July 1992

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 1992

ABSTRACT

The purpose of this paper is two-fold. First, it presents a simple, recursive, polynomial-time heuristic for finding the largest clique in a graph. The number of levels of recursion are limited. Computational results on a wide variety of both random and practical test problems with up to 587 nodes indicates that the heuristic is fast and generally optimal.

Secondly, when we remove the limit on the recursion levels, the algorithm will find the optimal solution. For uniform random graphs with edge density less than 50%, this algorithm is expected to be exponential in half the log of number of vertices, i.e. $\left(O\left(\sqrt{\log n}\, n^{\lfloor \log n+3 \rfloor/2}\right)\right)$. We generalize this result to virtually all other Branch and Bound algorithms for solving the maximum clique (maximum independent set) problem. Moreover, this bound is conservative. Our computational results provide some empirical evidence to support this statement.

Keywords: graph theory; maximum clique; independent set; heuristic.

RÉSUMÉ

Dans cet article, nous présentons, dans un premier temps, une heuristique simple en temps polynomial pour trouver une clique maximale dans un graphe. Cette heuristique est récursive, mais le nombre de niveaux de récursion est limité. Des résultats numériques pour un grand ensemble de problèmes tests, tant aléatoires que tirés d'applications réelles, possédant jusqu'à 587 sommets, montrent que l'heuristique est rapide et en général optimale.

Dans un deuxième temps, en éliminant la limite sur le nombre de niveaux de récursion, nous obtenons un algorithme exact. Pour les graphes aléatoires uniformes, cet algorithme possède une complexité espérée qui est bornée, de façon conservatrice, par une fonction exponentielle du logarithme du nombre de sommets $\left(O\left(\sqrt{\log n}\, n^{\lfloor \log n+3 \rfloor/2}\right)\right)$. Nous généralisons ensuite ce résultat à la plupart des autres algorithmes énumératifs pour le problème de clique maximale (ou, de façon équivalente, pour le problème d'ensemble stable maximal). Nos résultats numériques apportent une confirmation empirique partielle de ces conclusions.

Mots clés : théorie des graphes; clique maximale; ensemble stable; heuristique.

1. INTRODUCTION

A clique is a subgraph of a finite graph such that every pair of vertices is connected by an edge. The problem of finding the largest clique, or equivalently, the largest complete subgraph is well-known to be NP-hard (Garey & Johnson [1979]). Therefore, it is unlikely that a polynomial algorithm can be developed for solving this problem. Currently, the best theoretical result is due to Robson [1986] who presents an optimal algorithm which requires $O(2^{0.276n})$ operations in the worst case.

Until recently, the majority of the optimal algorithms in the literature (Bron & Kerbosch [1973], Corneil & Graham [1973], Loukakis & Tsouros [1982], and Tarjan & Trojanowski [1977]) find *all* maximal cliques in a graph (i.e., all cliques such that no new vertex can be added to obtain a larger clique). Clearly, this includes all of the largest cliques. They are all computationally impractical for graphs with more than about 100 vertices since the CPU requirements grow exponentially.

A number of papers have appeared more recently (Balas & Yu [1986], Robson [1986], Gendreau, Picard & Zubieta [1988], Friden, Hertz & de Werra [1990], Carraghan & Pardalos [1990] and Balas & Xue [1991]) that use implicit enumeration schemes to find the (single) largest clique in a graph. Balas & Yu have successfully found the optimal solution for random problems with up to 400 vertices and 30,000 edges (40% edge density) in about 9 minutes of CPU time on a VAX 11/780. Babel & Tinhofer [1990] present another branch and bound approach that generally outperformed Balas & Yu on a large number of randomly generated problems. The other algorithms are competitive. Friden, Hertz & de Werra solved one random problem with 1000 vertices and 30 % density, but it took 5 hours of CPU time on a VAX Station 11/RC.

Several (simple) heuristics have been suggested for finding a large clique in a graph, but until recently, very little attention has been paid to the computational performance of these methods, since in most instances, the clique finding routines were a secondary concern to the authors. In the last few years, a number of new heuristics such as those by Friden, Hertz & de Werra [1989] and Gendreau, Soriano & Salvail [1992] have been developed based on Tabu Search techniques, and have been tested successfully on random problems with up to 500 vertices.

The problem of finding the largest clique in a graph has a number of important practical applications. Auguston and Minker [1970], among others, use cliques as a basis for cluster analysis in a graph. Gotlieb and Kumar [1968] use cliques in a graph information model where related pieces of information are connected by an edge. Other applications occur in the generation of a thesaurus (Meethan [1966]) and the minimization of electrical cable length (Abraham [1965]).

Salazar and Oakford [1974] introduced the application of largest cliques in the timetabling problem. If each course is a vertex and an edge represents two courses in conflict, then the largest clique corresponds to a set of courses which must be assigned to different periods. The largest clique gives a lower bound on the number of colours required to colour the vertices of the graph.

Salazar and Oakford presented a simple heuristic for finding a large clique. Their approach, described in Section 2, is extended in Section 3 by adding one level of recursion. The resulting algorithm is bounded by $O(n^3)$ operations. Whereas the basic algorithm rarely finds the largest clique, adding one level of recursion provides a fairly reliable heuristic. Carter, Laporte & Chinneck [1992] have incorporated the heuristic described in this paper in a program for examination timetabling that has been implemented at four different universities in three countries.

In Section 4, the method is extended to an optimal algorithm which has an expected number of operations bounded by:

$$O(\sqrt{\log n}\, n^{\lfloor \log n + 3 \rfloor / 2})$$

This behaviour is illustrated on a wide variety of test problems with up to 587 vertices in Section 5.

2. BASIC ALGORITHM

Salazar and Oakford [1974] proposed a simple heuristic for finding a large clique in a graph. It is based on the premise that the vertex of minimum degree in a graph is the least likely candidate for inclusion in a large clique. The algorithm iteratively deletes the vertex of minimum degree until the remaining subgraph is itself a clique. More precisely, when the vertex of minimum degree is connected to all $(n-1)$ remaining vertices, the current vertex set clearly forms a clique. The algorithm also provides an upper bound on the size of the maximum clique. Observe that, when we delete a vertex v_p we could be deleting

a vertex in a clique of size $[deg(v_i)+1]$. Hence the maximum degree plus one of all deleted vertices is an upper bound on the maximum clique that might have been deleted.

The heuristic is summarized below.

CLIQUE0 (*G, CMIN, CMAX, CLIQUE*): Given a graph $G = (V,E)$, the routine computes *CMIN* and *CMAX* and stores the largest clique found in the vector *CLIQUE*.

Step 0: Set $CMAX = 0$, $n = |V|$.

Step 1: Let v_i be the vertex of minimum degree, $deg(v_i)$, in the graph G.

Step 2: If $deg(v_i) = (n-1)$, go to step 5.

Step 3: If $deg(v_i)+1 > CMAX$, set $CMAX = deg(v_i) + 1$.

Step 4: Remove node v_i and all of its incident edges from the graph. Let $n = n-1$ and go to step 1.

Step 5: The current subgraph is a clique. All vertices have degree $(n-1)$. Let $CMIN = n$, store the remaining n vertices in the vector *CLIQUE* and stop.

On completion *CMIN* is the size of the largest clique found and *CMAX* is an upper bound on the largest clique which could have been missed.

On a number of actual timetabling problems, this algorithm tends to find a clique which is close to the actual maximum size, but the bound *CMAX* is usually considerably overstated. For example, on a timetable problem with 121 vertices and a density of 38%, *CMIN* is 17, *CMAX* is 32 and the actual maximum clique has 19 vertices. Several other examples are presented in the computational results section.

The number of operations depends on the storage structure used for the graph. In the current implementation, the graph is stored as a bit matrix and the number of operations is bounded by $O(n^2)$ where n is the original number of vertices.

3. RECURSIVE ALGORITHM

The recursive version of the algorithm was designed to improve the upper bound estimate *CMAX*. Consider step 3 of the basic algorithm. When a vertex of degree $deg(v_i)$ is deleted, we may be deleting a vertex in a clique of size $[deg(v_i)+1]$. But this is only true if all of the vertices adjacent to vertex v_i form a clique.

Consider the induced subgraph G_i formed by taking all vertices adjacent to v_i at step 3 and their associated edges. Suppose we apply the algorithm 'CLIQUE0' to subgraph G_i. The algorithm will return a clique of size $CMIN_i$ and an upper bound of $CMAX_i$. Therefore, we have discovered a clique of size $[CMIN_i+1]$ (by including vertex v_i) and, deleting v_i could delete a clique of size $[CMAX_i+1]$ in the worst case.

CLIQUE1 (*G, CMIN, CMAX, CLIQUE*): Given a graph $G = (V,E)$, the routine computes *CMIN* and *CMAX* and stores the largest clique found in the vector *CLIQUE*.

Step 0: Let $CMAX = 0$, $CMIN = 0$, $n = |V|$.

Step 1: Let v_i be the vertex of minimum degree, $deg(v_i)$, in the graph G.

Step 2: If $n \leq CMIN$, stop. If $deg(v_i) = (n-1)$, go to step 5.

Step 3: If $deg(v_i)+1 \leq CMIN$, then go to step 4. Otherwise, let V_i be the set of all vertices adjacent to v_i. Let G_i be the graph induced on G by the vertices V_i.

Call CLIQUE0 (G_i, $CMIN_i$, $CMAX_i$, $CLIQUE_i$)

If $CMIN_i+1 > CMIN$, set $CMIN = CMIN_i+1$ (and save the new largest clique, the vector $CLIQUE_i$ plus v_i in a vector *CLIQUE*). If $CMAX_i+1 > CMAX$, set $CMAX=CMAX_i+1$. If $CMIN > CMAX$, set $CMAX=CMIN$.

Step 4: Remove node v_i from the graph G, set $n = n-1$ and go to step 1.

Step 5: The remaining n vertices form a clique. Set $CMIN = n$, store the vertices in the vector *CLIQUE* and stop.

On completion, *CMIN* is the size of the largest clique found and *CMAX* is an upper bound on the largest clique which may have been lost.

Consider the following simple estimate of the expected number of operations. The operation count for the algorithm is now dominated by step 3. Except for pathological examples, the expected degree of each vertex deleted will be bounded above by $O(pn)$ where p is the edge density of the graph. Therefore, each iteration of step 3 is bounded by $O(p^2n^2)$. Step 3 will call the subroutine $O(n)$ times for an overall bound of $O(p^2n^3)$ operations. In other words, the recursive version requires $O(p^2n)$ times as many operations as the basic algorithm. This estimate will normally be high since recursion will not be required for every vertex. To derive a worst case bound, observe that the degree of each induced subgraph is always

bounded by n. Hence the algorithm requires at most $O(n^3)$ operations.

The effect on the 121 vertex problem was dramatic. The upper bound CMAX was reduced from 32 to 21. More surprising, however, was the fact that the induced problem in step 3 actually found the optimal clique of size 19.

4. OPTIMAL ALGORITHM

Routine CLIQUE1 is easily converted into an optimal procedure by simply making it totally recursive. At step 3 of CLIQUE1, we call CLIQUE1 instead of CLIQUE0. The recursion will automatically continue until either the bound on the largest clique is less than *CMIN*, or the induced subgraph is itself a clique. The values of *CMAX* and *CMIN* will always be equal and each occurrence of *CMAX* can be replaced by *CMIN*.

It is easy to see that this algorithm is precisely equivalent to a simple implicit enumeration scheme, i.e., at each stage, we select a vertex and solve one problem with the vertex included (plus its neighbours), and solve a second problem with the vertex deleted. It is not our intention to suggest that this approach is competitive with Babel & Tinhoffer [1990], Friden, Hertz & de Werra [1990] or Balas & Xue [1991] for example. They all do considerably more work on bounding and reducing the size of each subproblem. However, our simple optimal routine led to a very interesting theoretical result which we were able to then extend to other optimal algorithms.

Consider the expected behaviour of the algorithm on a random graph $G(n,p)$ with n vertices where each edge in the graph is included in the graph with probability p.

Theorem: For any fixed value of $p \leq 0.5$, the *expected* complexity of the optimal algorithm is bounded by:

$$O\left(\sqrt{\log n}\, n^{\lceil \log n + 3 \rceil / 2}\right)$$

(Note that $\sqrt{\log n}$ grows very slowly.)

We will prove this theorem using a sequence of lemmas.

Lemma 1: In a uniform random graph with n vertices and density p, the expected number of complete subgraphs of order k is given by:

$$E\left[N_k^C\right] = \binom{n}{k} p^{\frac{k(k-1)}{2}}$$

Proof: The expression simply counts the number of ways of picking k vertices times the probability that a random choice is complete. •

Lemma 2: The total number of operations for a subproblem with m vertices at level k is bounded by αm^2 for some constant α. (This does *not* include operations at higher levels of recursion.)

Proof: Consider the steps involved in algorithm CLIQUE1. The initialization step requires $O(m^2)$ operations to compute the degree of each vertex. The remaining steps 1-5 are executed m times, once for each vertex.

Step 1 requires m operations to determine the vertex of minimum degree. Step 3 requires m operations to construct the index set, V_i. (It is not necessary to actually construct the graph G_i. We simply use V_i to index the elements of the original matrix G.) Finally, step 4 requires $O(m)$ operations to update the index set V and the degrees of the remaining vertices. Therefore, the overall operation count is bounded by αm^2 for some constant α. •

Lemma 3: Suppose that the number of operations required to solve a subproblem with m_i vertices is bounded by some general function $\phi(m_i)$. We will assume that $\phi(m_i)/m_i$ is non-decreasing. (This is equivalent to saying that each subproblem requires *at least* a linear amount of time to process.) The expected number of operations required to solve all of the subproblems at level k is bounded by:

$$B_k = \phi(n) n^k p^{k(k+1)/2}$$

Proof: Observe that, if the i-th subproblem has m_i vertices, then the total number of operations required to solve all level k subproblems is:

$$T_k \le \sum_i \phi(m_i)$$

Suppose that the total number of vertices in all subproblems at level k is equal to L_k. Hence,

$$\sum_i m_i = L_k$$

It is easy to see that:

$$T_k \le \sum_i m_i \frac{\phi(m_i)}{m_i} \le \sum_i m_i \frac{\phi(n)}{n} = L_k \frac{\phi(n)}{n}$$

Observe that every vertex in any subproblem at level k corresponds directly to a unique complete subgraph of size $k+1$ in the original graph, i.e., the k vertices at each previous level all form a complete subgraph of size $k+1$ with each vertex in a subproblem.

Hence the expected total number of vertices in all subproblems at level k, $E[L_k]$, is bounded by $E[N^C_{k+1}]$, the expected number of complete subgraphs of size $k+1$. The enumeration scheme ensures that no complete subgraphs will be repeated.

We wish to determine a bound on the expected number of operations.

$$E[T_k] \le E\left[L_k \frac{\phi(n)}{n}\right] = E[L_k] \frac{\phi(n)}{n} \le E[N^C_{k+1}] \frac{\phi(n)}{n} = \binom{n}{k+1} p^{k(k+1)/2} \frac{\phi(n)}{n}$$

Observe that:

$$\binom{n}{k+1} < n^{(k+1)}$$

Therefore;

$$E[T_k] \le \phi(n) n^k p^{k(k+1)/2} = B_k$$

Lemma 4: The bound on the expected number of operations attains its maximum at:

$$B_{k^*} = \phi(n)\, p^{-\frac{1}{8}} n^{\lfloor \log n - 1 \rfloor / 2}$$

where:

$$k^* = \log n - \frac{1}{2}$$

In order to simplify the notation, we will use $\log n$ to mean $\log_b n$ where $b = \left(\frac{1}{p}\right)$. Observe that this function is truly logarithmic whenever $p \leq 0.5$. Note that, in general, k^* will not be integer.

Proof: The maximum value of B_k corresponds to the maximum of:

$$\ln B_k = k \ln n + \frac{(k+1)k}{2} \ln p + \ln \phi(n)$$

Differentiating with respect to k gives a maximum when:

$$\ln n + \left(k + \frac{1}{2}\right) \ln p = 0$$

$$\left(k + \frac{1}{2}\right) = \frac{\ln n}{\ln \frac{1}{p}}$$

$$k^* = \log n - \frac{1}{2}$$

Now,

$$B_{k^*} = \phi(n)\, n^{k^*}\, p^{k^*(k^*+1)/2}$$

Note that:

$$p^{k^*(k^*+1)/2} = p^{(k^*+\frac{1}{2})^2/2}\, p^{-\frac{1}{8}}$$

and

$$n^{k^*} = n^{-\frac{1}{2}}\, n^{(k^*+\frac{1}{2})}$$

and therefore,

$$B_{k^*} = \phi(n)\, n^{-\frac{1}{2}} \left[n\, p^{(k^*+\frac{1}{2})/2}\right]^{(k^*+\frac{1}{2})}\, p^{-\frac{1}{8}}$$

But, since $p^{(\log n)} = \frac{1}{n}$, we get,

$$n\,p^{(k^*+\frac{1}{2})/2} = n\,p^{\lceil \log n \rceil/2} = n^{\frac{1}{2}}$$

and therefore:

$$B_{k^*} = \phi(n)\,p^{-\frac{1}{4}}\,n^{\lceil \log n - 1 \rceil/2}$$

as required. •

Lemma 5: Finally, the total number of operations is bounded by:

$$O\left(\sqrt{\log n}\,\phi(n)\,n^{\lceil \log n - 1 \rceil/2}\right)$$

Proof: Let B denote the bound on the total expected number of operations defined by:

$$B = \sum_{k=0}^{n-1} B_k$$

Now,

$$\frac{B_k}{B_{k^*}} = n^{k-k^*}\,p^{\lceil (k+1)k-(k^*+1)k^* \rceil/2} = \left[n\,p^{(k^*+\frac{1}{2})}\right]^{(k-k^*)}\,p^{(k-k^*)^2/2}$$

But, $p^{k^*+\frac{1}{2}} = p^{\log n} = \frac{1}{n}$. Therefore,

$$\frac{B_k}{B_{k^*}} = p^{(k-k^*)^2/2}$$

And we may write:

$$B = B_{k^*} \sum_{k=0}^{n-1} p^{(k-k^*)^2/2}$$

Consider the two complementary sets:

$$K = \left\{k \in \{0,1,\ldots,n-1\} \mid (k-k^*)^2 \geq 2k^*+1\right\}$$

and

$$\overline{K} = \left\{k \in \{0,1,\ldots,n-1\} \mid (k-k^*)^2 < 2k^*+1\right\}$$

For $k \in K$,

$$p^{(k-k^*)^2/2} \le p^{k^*+\frac{1}{2}} = \frac{1}{n}$$

And, since $|\overline{K}| < n$:

$$\sum_{k \in \overline{K}} p^{(k-k^*)^2/2} < 1$$

For $k \in \overline{K}$, $p^{(k-k^*)^2/2} \le 1$ and therefore,

$$\sum_{k \in \overline{K}} p^{(k-k^*)^2/2} \le |\overline{K}|$$

Now, if $k \in \overline{K}$, then

$$|k-k^*| < \sqrt{2k^*+1} = \sqrt{2 \log n}$$

Therefore, *every* $k \in \overline{K}$ must lie in the range: $k^* \pm \sqrt{2 \log n}$ and, therefore,

$$|\overline{K}| \le 2\sqrt{2 \log n}$$

Hence, we can write:

$$B < (1 + 2\sqrt{2 \log n})\, B_{k^*}$$

or

$$B < (1 + 2\sqrt{2 \log n})\, \phi(n)\, p^{-\frac{1}{2}}\, n^{\lfloor \log n - 1 \rfloor/2}$$

and the result follows. •

The theorem follows by substituting $\phi(n) = \alpha n^2$.

When $p \le 0.5$, k^* has been shown to be a logarithmic function of the size of the problem. The next section supports this result on a variety of practical test problems. Observe that, when $p > .5$, k is still, technically speaking, a logarithmic function of n. However, the base b is less than 2 and the expected performance of the routine deteriorates rapidly as p increases.

The preceding proof (after substituting different definitions of $\phi(n)$) applies to most enumerative algorithms for the maximum clique problem. It may be necessary however to redefine the original search tree structure to conform with the assumptions of this proof. In particular, the k^{th} level of the tree is made up of all subproblems with precisely k vertices *forced* into the solution.

5. COMPUTATIONAL RESULTS

The algorithm was coded in PL/I and tested on a wide variety of problems on an IBM 4043 computer. One simple modification was made to the algorithm to limit the number of levels of recursion on any given run to a specified value called *LMAX*. With *LMAX* = 0, we get the basic non-recursive algorithm, CLIQUE0.

The first set of test problems were random graphs of the form $G(n,p)$, where n is the number of vertices and the edges are generated randomly with probability p. The resulting graph will have an edge density close to p.

Table 1 and Table 2 summarize the results for various values of n with $p = 0.5$ and $p = 0.2$ respectively. Execution times are given in CPU seconds. The k^* value represents the recursion level with the maximum expected number of operations. Therefore, we might expect that the number of operations would increase exponentially up to level k^* and then drop off quickly. Each problem was solved using increasing levels of recursion (LMAX) until *CMIN* = *CMAX* (i.e., until optimality was verified).

Table 1: G(n,.5)

	Level	CMIN	CMAX	CPU
n=50	0	6	19	.070
k^*=5.14	1	7	9	.613
	2	7	7	.903
n=100	0	9	41	.257
k^*=6.14	1	9	19	5.117
	2	9	11	16.856
	3	9	9	18.089
n=150	0	7	63	.567
k^*=6.73	1	10	29	18.126
	2	10	15	90.018
	3	10	10	136.271

Table 2: G(n,.2)

	Level	CMIN	CMAX	CPU
n=50	0	4	8	.067
k^*=1.93	1	4	4	.213
n=100	0	5	15	.257
k^*=2.36	1	5	5	1.150
n=150	0	4	23	.567
k^*=2.61	1	5	6	3.460
	2	5	5	3.503
n=200	0	4	30	.983
k^*=2.79	1	6	7	7.480
	2	6	6	8.220

Several conclusions can be drawn from the results. In fact, the maximum number of levels of recursion required to find the optimal solution is consistently well below k^*. One can also see the pronounced effect of density on the CPU time required. In particular the graph $G(150,0.5)$ required over 2 minutes of CPU time to solve to optimality while $G(150,0.2)$ took only 3.5 seconds.

Perhaps the most surprising result is that all problems actually found a largest clique with only one level of recursion. Higher levels were required only for confirmation.

Since the largest clique algorithm was originally developed for use in student timetable problems, the algorithm was next tested on a variety of actual conflict graphs. The first three problems came from the North York Board of Education. (The city of North York is part of Metropolitan Toronto.) In each of these schools, students submit "course request" lists at the end of each school year as input to the timetable development process for the next year. A conflict graph was constructed for each school where the vertices represent courses and an edge connects two vertices if any student wishes to take that pair of courses. The difficulty of the problem is in direct proportion to the degree of flexibility allowed for student requests.

NY336 is a junior high school covering grades 7, 8 and 9. Students are on a year system (i.e., all courses are for a full year). The number of required courses per student is relatively high. The resulting graph is composed of separable clusters of vertices.

NY415 is a senior high school (grades 10 to 13) which is also on a year system. There are more elective courses, but grade 10 students tend to take grade 10 courses, etc.

NY430 is a senior high school on the semester system. The school year is divided into two semesters and students normally take three or four different courses each semester. A nominal grade 12 student could select courses from all grade levels. The results for the three schools are summarized in Table 3.

Table 3: North York Schools

	Level	CMIN	CMAX	CPU
NY336	0	12	19	.477
n=139	1	14	18	.967
p=.145	2	14	17	1.193
k^*=2.06	3	14	16	1.347
	4	14	15	1.430
	5	14	14	1.423
NY415	0	13	37	.866
n=181	1	18	26	9.399
p=.289	2	18	22	19.409
k^*=3.69	3	18	20	26.282
	4	18	19	27.275
	5	18	18	27.501
NY430	0	19	38	.927
n=190	1	20	28	9.429
p=.267	2	21	22	25.965
k^*=3.47	3	21	21	28.578

Clearly, these results are quite different from the behaviour exhibited by the random graph models. In these problems, $k^* = \log_b n - \frac{1}{2}$, is below the actual number of levels required. However, observe that the additional CPU time required to solve the problems above recursion level k^* does not tend to increase

dramatically with each level as could be inferred by the definition of k^*. For example, in NY336, the predicted maximum level was 2, but the optimal level 5 was not significantly more expensive. CPU time appears to grow exponentially up to level k^*. For the NY336 problem, we observe that five levels of recursion is faster than four levels. This is probably due to the fact that a higher level of recursion allows us to find better bounds sooner, resulting in a smaller branching tree. (The same phenomenon, more pronounced, occurs later in table 5 with n=200.)

The two most important results of this experiment are that one level of recursion was sufficient to find the largest clique in two of the schools. A second level was required for NY430. And, finally, the algorithm solved three practical problems in, at most, 28 seconds of CPU time.

The next test problem was based on an examination timetabling problem at the University of Waterloo from the fall term of 1982. As before, vertices represent courses and an edge connects two vertices if any student is required to write both examinations. The results are summarized in Table 4.

Table 4: Waterloo Examination Timetable

	Level	CMIN	CMAX	CPU
n=587	0	20	45	8.499
p=.087	1	22	30	38.764
k^*=2.11	2	22	25	89.671
	3	22	23	103.060
	4	22	22	104.193

Even though this problem is large compared to the high school data, the results are similar. The actual largest clique was found after one level of recursion. The exponential growth of CPU time did not continue beyond the predicted maximum level k^*=2. And, most important, a rather large practical problem was solved to optimality in less than two minutes.

The final set of test problems are called simple "Timetable Random Graphs". They represent a first attempt to construct random problems which are more representative of timetabling problems than the $G(n,p)$ model.

These graphs add a third property to the model as $T(n,p,c)$ where n and p are the size and density of the graph. The parameter c corresponds to the number of courses per student. A random graph is constructed by selecting c vertices at random and connecting all pairs by edges, thus simulating a student selecting c courses. The number of such selections that must be made to produce a graph with expected density p can be computed analytically. The results on a number of test problems is given in Table 5, Table 6, and Table 7.

Table 5: 'Timetable Random Graphs: c=5'

	Level	CMIN	CMAX	CPU
n=50	0	6	19	.070
p=.5	1	7	9	.613
k^*=5.14	2	7	7	.903
n=100	0	8	39	.260
p=.5	1	10	19	5.040
k^*=6.14	2	10	12	18.085
	3	10	10	19.682
n=150	0	10	60	.567
p=.5	1	11	28	17.089
k^*=6.73	2	11	15	95.171
	3	11	11	142.730
n=200	0	11	79	1.003
p=.5	1	12	37	39.830
k^*=7.14	2	12	20	290.407
	3	12	14	731.759
	4	12	12	716.223

All of these problems actually found the largest clique after only one level of recursion. The largest problem, $T(200, .5, 5)$, required less than 40 seconds of CPU time at level one.

Table 6: 'Timetable Random Graphs: c=10'

	Level	CMIN	CMAX	CPU
n=50	0	12	20	.070
p=.5	1	13	14	.530
k^*=5.14	2	13	13	.613
n=100	0	14	36	.253
p=.5	1	14	22	4.676
k^*=6.14	2	14	16	18.275
	3	14	14	19.215
n=150	0	11	55	.567
p=.5	1	14	31	16.426
k^*=6.73	2	14	21	109.410
	3	14	15	237.871

Table 7: 'Timetable Random Graphs: c=15'

	Level	CMIN	CMAX	CPU
n=50	0	16	21	.063
p=.5	1	16	16	.433
k^*=5.14				
n=100	0	15	38	.253
p=.5	1	16	23	4.873
k^*=6.14	2	16	17	18.519
	3	16	16	18.232
n=150	0	15	53	.563
p=.5	1	16	32	15.546
k^*=6.73	2	16	23	104.670
	3	16	19	224.818
	4	16	17	225.875
	5	16	16	224.385

6. CONCLUSIONS AND EXTENSIONS

From a practical perspective, the heuristic obtained by limiting the algorithm to one level of recursion provides a fast and generally optimal solution. The number of operations is conservatively bounded by $O(p^2n^3)$. The test problems empirically substantiate this result and all problems were solved in less than 40 seconds of CPU time. This is particularly useful since many applications such as cluster analysis do not require the actual largest clique. A good approximation is often sufficient.

For the randomly generated problems the bound of $O(p^3n^4)$ on total operations with two levels of recursion is consistent with the observed computation times. This bound relies on the independence assumption of the edges in a random graph. Clearly, this assumption is not valid for actual school data.

For the practical problems, the assumption that the first level induced subgraph has $O(pn)$ vertices is reasonable. (Note that *CMAX* at level 0 is equal to the maximum induced subgraph at level 1 which is less than pn for all test problems.) However, level 2 subgraphs contain more than $O(p^2n)$ vertices since the vertices in the level 1 problem are, by definition, related.

In the worst case, level 2 problems will also have $O(pn)$ vertices. Using this estimate, (and the procedure outlined in section 4), we obtain an upper bound on the number of operations as $O(n(pn)(pn)^2)$ or $O(p^3n^4)$ which produced a reasonable fit with the actual results. The largest practical problem was solved in 90 seconds. The largest random problem required 200. In all cases, two levels of recursion produced the optimal solution.

For the optimal algorithm, we showed that the expected number of operations can be bounded by a function $O(\sqrt{\log n}\, n^{\lfloor \log n + 3 \rfloor/2})$. Furthermore, the recursion level k^* for which the bound on the number of operations is maximal is a logarithmic function of n. For the random test problems, fewer levels were often adequate to determine the solution. For the practical cases, although the number of levels often exceeds k^*, the increase in CPU time does not continue to grow exponentially beyond level k^*, as predicted.

The difference in behaviour observed between random and practical problems is difficult to justify theoretically. However, a partial explanation can be found by considering the structure of practical

graphs. The vertex degrees of random problems tend to lie close to the mean value pn (Cornuéjols [1978]). Practical problems on the other hand exhibit a much wider variation. For example, in the Waterloo problem (n=587), 152 vertices are degree 20 or less and 100 vertices have degree 90 or more. In addition, the cliques tend to be much larger than for random graphs. Consequently, the majority of vertices are eliminated at early levels. Hence, only a relatively few subproblems require higher levels of induction.

One possible improvement to the current algorithm would involve a more precise estimate for the value of *CMAX*. Selkow [1978] for example gives an upper bound for *CMAX* based on bounds for the clique size in the induced subgraphs. A tighter upper bound will certainly reduce the number of levels of recursion which would be required for optimality. However, tighter bounds would also increase the time required per iteration. The net effect of this trade-off is not obvious and we leave the question as a topic for future research.

The worst case performance of the algorithm with j levels of recursion for any fixed j is arbitrarily bad. Consider the following example based on a variation of a graph given by Johnson [1974]. The problem is composed of a clique of size k and, for every subset of j vertices in the clique, we construct a complete bipartite subgraph on $2k$ vertices as shown in Figure 1. Each vertex in the j-subclique is connected to every vertex in the bipartite subgraph.

There are $\binom{k}{j}$ bipartite subgraphs which, for $k>>j$ is approximately k^j. Finally, we connect all P vertices to all Q vertices between each pair of subgraphs. The degree of each vertex in the clique k is given by:

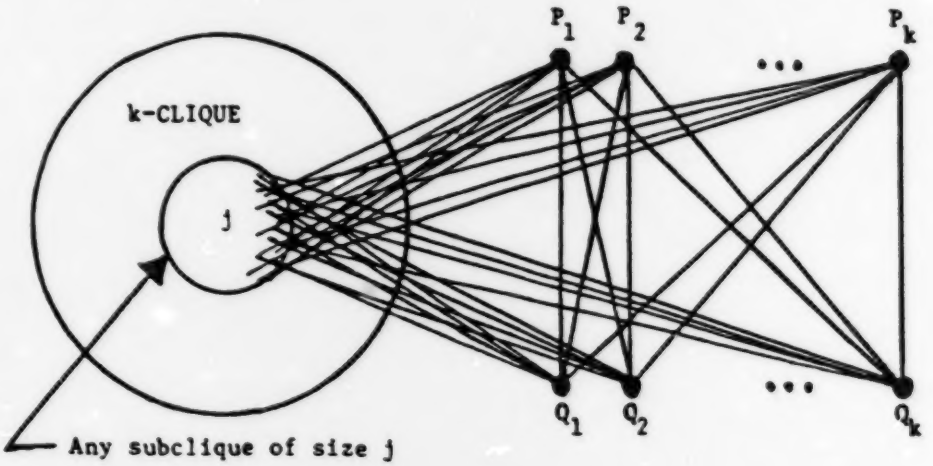
$$(k-1) + 2k\binom{k-1}{j-1} = 2k^j$$

The degree of each vertex in P or Q is:

$$k\binom{k}{j} = k^{j+1}.$$

Hence, the recursive algorithm with j levels of recursion will delete the nodes in the clique first. At level j, the induced subgraph will appear as in Figure 1 without the nodes in j and the procedure will find a

Figure 1: Graph G with Relative Error $k/(j+2)$



largest clique of size $(j+2)$. Hence, for any fixed j, the relative error $\frac{k}{(j+2)}$ is unbounded. Moreover, the edge density of the graph is close to 50%.

The example is not without redeeming features for the present algorithm. The size of the graph is $\approx kj$ and $(j+1)$ levels of recursion are sufficient to find and verify the optimal solution. Therefore, even for this pathological example, the number of levels required for optimality is *still* a logarithmic function of the size of the graph.

7. ACKNOWLEDGEMENTS

We would like to thank Derek Corneil, Jeffrey Sidney, Art Warburton, Bill Pulleyblank, John Buzacott and two anonymous referees for their many helpful comments and suggestions.

This research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

References

1. Abraham, C.T., "An Application of Clustering Techniques to Minimizing the Number of Interconnections in Electrical Assemblies" in *Some Problems in Information Science*, (ed.: M. Kochen), The Scarecrow Press, New York, 1965, pp. 252-264.
2. Auguston, J.G., and Minker, J., "An Analysis of Some Graph Theoretical Cluster Techniques", *J ACM, 17*, No. 4, Oct. 1970, pp. 571-588.
3. Babel, L. and Tinhofer, G., "A Branch and Bound Algorithm for the Maximum Clique Problem", *ZOR - Methods and Models of Operations Research 34*, 1990, pp. 207-217.
4. Balas, E. and Yu, C.S., "Finding a Maximal Clique in an Arbitrary Graph", *SIAM J. Computing. 15*, No. 4, Nov. 1986, pp. 1054-1068.
5. Balas, E. and Xue, J., "Minimum Weighted Coloring of Triangulated Graphs, With Application to Maximum Weight Vertex Packing and Clique Finding in Arbitrary Graphs", *SIAM J. of Computing 20*, No. 2, pp. 209-221, 1991.
6. Bron, C. and Kerbosch, J., "Algorithm 457: Finding All Cliques of an Undirected Graph", *Communications of the ACM, 16*, No. 9, Sept. 1973, pp. 575-577.
7. Carraghan, R. and Pardalos, M., "An Exact Algorithm for the Maximum Clique Problem" *Operations Research Letters 9*, 1990, pp. 375-382.
8. Carter, M.W., Laporte, G. and Chinneck, J.W., "A General Examination Scheduling System", Working paper #92-03, Industrial Engineering, University of Toronto, March 1992.
9. Corneil, D.G. and Graham, B., "An Algorithm for Determining the Chromatic Number of a Graph", *SIAM J. of Computing 2*, pp. 311-318, 1973.

10. Cornuéjols, G., "Degree Sequences of Random Graphs", Discussion Paper 7818, C.O.R.E., Université Catholique de Louvain, Belgium, May 1978.

11. Friden, C., Hertz, A. and de Werra, D., "Stabulus: A Technique for Finding Stable Sets in Large Graphs with Tabu Search", *COMPUTING 42*, 1989, pp 35-44.

12. Friden, C., Hertz, A. and de Werra, D., "TABARIS: An Exact Algorithm Based on Tabu Search for Finding a Maximum Independent Set in a Graph", *Computers and Operations Research 17*, No. 5, 1990, pp. 437-445.

13. Garey, M., and Johnson, D., *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman & Co., San Francisco, 1979.

14. Gendreau, M., Picard, J.-C. and Zubieta, L., "An Efficient Implicit Enumeration Algorithm for the Maximum Clique Problem" in *Advances in Optimization and Control*, (H.A. Eiselt and G. Pederzoli, Eds.), Springer-Verlag, Berlin, 1988.

15. Gendreau, M., Soriano, P. and Salvail, L., "Solving the Maximum Clique Problem Using a Tabu Search Approach", Centre de recherche sur les transports - Publication # 675, Université de Montréal, Montréal, Canada, 1992, to appear in *Annals of Operations Research: Special Issue on Tabu Search*.

16. Gotlieb, C.C., and Kumar, K. "Semantic Clustering of Index Terms", *J.ACM, 15*, No. 4, Oct. 1968, pp. 493-513.

17. Johnson, D. S., "Approximation Algorithms for Combinatorial Problems", *Journal of Computer and System Science 9*, 1974, pp. 256-278.

18. Loukakis, E. and Tsouros, C., "Determining the Number of Internal Stability of a Graph", *International J. of Computer Mathematics 11*, pp. 232-248, 1982.

19. Meethan, A.R., "Graph Separability and Word Grouping", *Proc. 21st Nat. Conf. ACM*, ACM Pub. P-66, Thompson Book Co., Washington D.C., 1966, pp. 513-514.

20. Robson, J.M., "Algorithms for Maximum Independent Sets", *Journal of Algorithms 7*, 1986, pp. 425-440.

21. Salazar, A. and Oakford, R.V., "A Graph Formulation of a School Scheduling Algorithm", *Communications of the ACM, 17*, No. 12, 1974, pp. 696-698.

22. Selkow, S.M., "New Bounds for the Clique Number of a Graph", *Information Processing Letters 7*, June 1978, pp. 173-174.

23. Tarjan, R.E., and Trojanowski, A.E., "Finding a Maximum Independent Set", *SIAM J. on Computing 6*, No. 3, Sept. 1977, pp. 537-546.